Data exchanges between the Smart Burst Buffer and the Resource Manager

1. Introduction

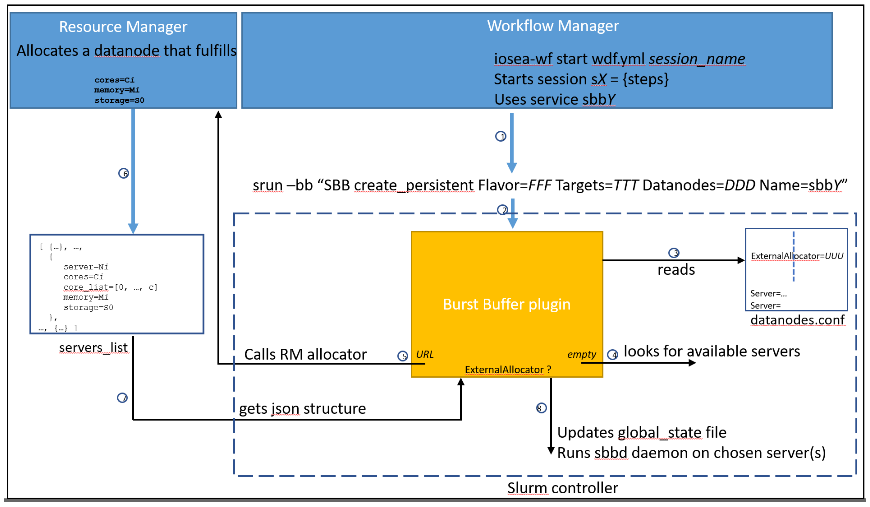
When the workload manager starts a workflow session (iosea-wf start wdf.yml <session name>), it starts all the ephemeral services that will be needed by any step included in that workflow.
Ephemeral services are currently restricted to Smart Burst Buffers (SBB).

Starting an ephemeral service consists in generating a slurm command that will create a persistent burst buffer.
The term "persistent" is used here in the context of flash accelerators and it means that the Burst Buffer(BB) will survive to the end of the job that created it. It will be usable by any job inside the same workflow session. It will be destroyed at the end of the workflow session, when no step uses it anymore.
The BB is said persistent within the session and ephemeral across sessions.

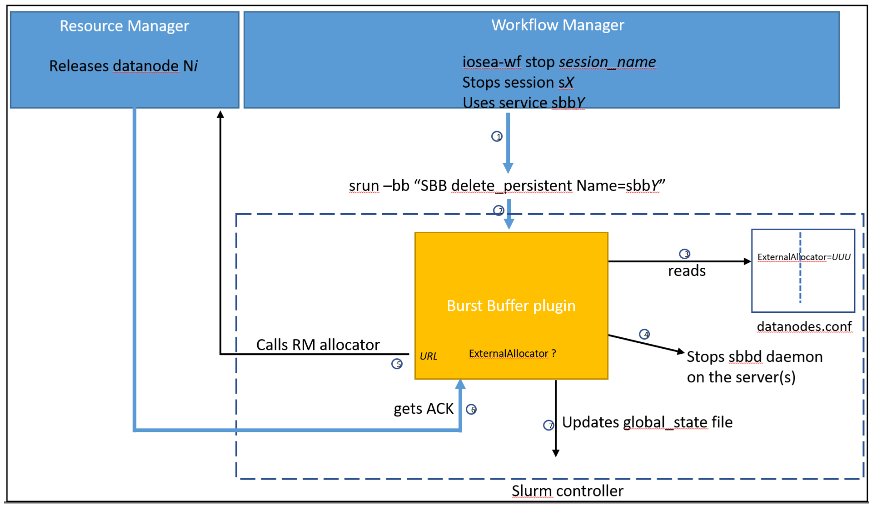
When the persistent BB creation slurm command is run, the BB plugin is activated. It detects that an external allocator(RM) is needed by analyzing the FA configuration file. In that case, it asks the RM to allocate any datanode needed for that particular ephemeral service, sending him the associated resources requirements. Upon positive ACK, the BB plugin updates its accounting information and runs the SBB daemons on the nodes allocated by the RM.

When the workflow session is stopped (iosea-wf stop <session name>), any ephemeral service previously started in this workflow session should be stopped (after any step using it has stopped too). Similarly to session start, stopping an ephemeral service consists in generating a slurm command that will destroy a persistent burst buffer (previously created during workflow session startup).
When the persistent BB release slurm command is run, the BB plugin is activated. It detects that an external allocator is needed by analyzing the FA configuration file. In that case, it stops the SBB daemons on the nodes allocated by the RM, then asks the RM to release any resource allocated on the datanode(s) needed for that particular ephemeral service, sending him the allocation id (obtained in the allocation request response).
Upon positive ACK, the BB plugin updates its accounting information.

The following scheme summarizes the allocation process:



The following scheme summarizes the release process:



In this page, we describe what we would like to be discussed with IT4I who is in charge of providing a resource manager for the IO-SEA project.

2. Requirements

The Resource Manager (RM) will be in charge of allocating the set of resources required by SBB or any other ephemeral service.

Resources requirements are sent to the RM, the RM allocates the required resources and sends a response back to SBB (see External Allocator for SBB).

The requirements are expressed in terms of cores, memory size and storage size. Note that other resources requirements might be added to this list as new ephemeral services are supported.

Asking for resources allocations will be done through a REST API, sending the allocation requirements and receiving the allocation response in json format.

2.1. Allocation requirements

- number of servers to allocate resources on: **NSA**
- number of cores to allocate (evenly) on each server: **CCC**
- memory size to allocate (evenly) on each server: **MMM**
- one of the following:
  - storage size to allocate (evenly) on each server: **SSS** . This is how SBB or SBF work today
  - a global storage size to be distributed among all the NSA servers: **GSS**. This is how GBF works today

2.2. Allocation response:

- number of servers actually allocated: should be NSA
- For each allocated server:
  - server name: **SN**
  - number of cores actually allocated. $\mathbf{C}_i$
  - list of allocated core numbers: $\mathbf{L}_i$
  - memory size actually allocated: $\mathbf{M}_i$
  - storage size actually allocated: $\mathbf{S}_i$

2.3. Response requirements:

- for each i:
  - C_i >= CCC
  - length(L_i) = C_i
  - M_i >= MMM
  - SBB and SBF allocation policies: S_i >= SSS
- sum(all C_i) >= CCC * NSA
- sum(all M_i) >= MMM * NSA
- GBF allocation policy: sum(all S_i) >= GSS

2.4. Authentication

TBD

3. Methods used

3.1. POST method

ENDPOINT: xxx/v1.0.0/servers/allocation

CONSUMES: at least application/json (Content-Type request header)

REQUEST BODY: v1.0.0_servers_allocation

key	type	comment
"servers"	Integer	number of servers to allocate
"cores"	Integer	number of cores to allocate on each server. May be 0 (needed later for GBF)
"msize"	Integer	memory size to allocate in MB (on each server). May be 0 (needed later for GBF)
"ssize"	Integer	storage size to allocate in MB (on each server). Exclusive with "gssize".
"gssize"	Integer	Global storage size. This size will be evenly distributed among the allocated servers (to be used by GBF or other ephemeral distributed service). Exclusive with "ssize".

RETURN TYPE: v1.0.0_servers_allocation_response

key	type	comment
"id"	Integer	allocation id This id is not required by the WM, but might be needed by the RM if it maintains a kind of allocations database. It might be useful during allocation release. To be discussed. Note that the DELETE method is based on this id.
"servers"	Integer	number of allocated servers
"properties"	array[v1.0.0_servers_allocation_properties]	properties of each allocation

v1.0.0_servers_allocation_properties

key	type	comment
"name"	String	allocated server name
"cores"	Integer	number of cores allocated on this server. Only SBB expects this to be > 0
"core_list"	array[Integer]	list of the allocated core numbers Empty list means "no binding"
"msize"	Integer	memory size (in MB) allocated on this server. Only SBB expects this to be > 0
"ssize"	Integer	storage size (in MB) allocated on this server

see "response requirements" for details about the response requirements.

The properties of what has actually been allocated are needed for the following reasons:

1. The resources granularity might not be of 1 unit on the servers to be allocated. Say for example that only entire sockets can be allocated on a server, and that we are asking for less cores than the number of cores per socket. In that case, more cores than what was required are allocated. The only expectation of the response is that the allocated quantities are at least equal to what was asked for.
2. In the GBF mode (that is not supported yet, but that will be in a near future), the global storage size that is asked for is first divided by the number of wanted servers. But what is actually allocated on each server depends on what is available at the moment the allocation is done. We allocate min(*required*, *available*). Then the remaining required size is again divided by the remaining servers, and so on.

Thus we are almost sure all the servers won't have the same storage size allocated on them. At least the last server will have GSS % NSA KB of storage size allocated.

3.2. DELETE method

ENDPOINT: xxx/v1.0.0/allocation/{allocation_id} (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

key	type	comment
"id"	Integer	id of the allocation to release

RESPONSES:

code	comment
200	allocation successfully released
500	allocation not found

4. Errors management during allocation request (POST method)

see https://restfulapi.net/http-status-codes/ for a full list of HTTP status codes.

4.1. Success

status code = 201

4.2. Resources not available

TBDiscussed:

- (synchronous call). But we are limited by the server http timeout...
- asynch. Always replied with an ID. Then we will poll to check if this id is allocated ==> cancel if wait time > timeout to be defined.
  In that case, we will need 2 more methods:
  1. GET method - endpoint=xxx/v1.0.0/allocation/{allocation_id}
     This would reply with the status of the allocation (allocated / not)
  2. We could use the DELETE method described below to cancel the allocation request

Several possible status codes, depending on the fact that the call is synchronous oor asynchronous, to be discussed:

1. status code 102 - Processing: "Indicates that the server has received and is processing the request, but no response is available yet." (see https://restfulapi.net/http-status-codes/#1xx)
   In that case, SBB will wait until a response is available, but up to the http timeout!!!
2. status code 404 - Not Found: "Indicates that the server can not find the requested resource." (see https://restfulapi.net/http-status-codes/#4xx)
3. status code 409 - Conflict: "Indicates that the request could not be completed due to a conflict with the current state of the resource." (see https://restfulapi.net/http-status-codes/#4xx)
4. status code 500 - Internal Server Error: "indicates that the server encountered an unexpected condition that prevented it from fulfilling the request." (see https://restfulapi.net/http-status-codes/#5xx)